Oracle8*i*

Replication Management API Reference

Release 2 (8.1.6)

December 1999

A76958-01

ORACLE

Oracle8i Replication Management API Reference, Release 2 (8.1.6)

A76958-01

Primary Author: William Creekbaum and Randy Urbano

Graphic Artist: Valarie Moore

Contributors: Nimar Arora, Alan Downing, Al Demers, Ira Greenberg, Denis Goddard, Maria Pratt, Jim Stamos, Curt Elsbernd, Jairaj Galagali, Viswanathan Krishnamurthy, Jing Liu, Pat McElroy, Arvind Rajaram, Wayne Smith, Eric Vandevelde, Lik Wong

Contents

9 Data Dictionary Views

6

Conflict Resolution

This chapter illustrates how to define conflict resolution methods for your replicated environment. The following topics are discussed:

- Prepare for Conflict Resolution
- Create Conflict Resolution Methods for Update Conflicts
- Create Conflict Resolution Methods for Uniqueness Conflicts
- Create Conflict Avoidance Methods for Delete Conflicts
- Audit Successful Conflict Resolution

Prepare for Conflict Resolution

Though you may take great care in designing your database and front-end application to avoid conflicts that may arise between multiple sites in a replicated environment, you may not be able to completely eliminate the possibility of conflicts. One of the most important aspects of replication is to ensure data convergence at all sites participating in the replicated environment.

When data conflicts occur, you need a mechanism to ensure that the conflict is resolved in accordance with your business rules and that the data converges correctly at all sites.

Oracle replication lets you define a conflict resolution system for your database that resolves conflicts in accordance with your business rules. If you have a unique situation that Oracle's pre-built conflict resolution methods cannot resolve, you have the option of building and using your own conflict resolution methods.

> **See Also:** *Oracle8i Replication* for conceptual information about conflict resolution methods and detailed information about data convergence for each method.

Plan

Before you begin implementing conflict resolution methods for your replicated tables, analyze the data in your system to determine where the most conflicts may occur. For example, static data such as an employee number may change very infrequently and is not subject to a high occurrence of conflicts. An employee's customer assignments, however, may change often and would therefore be prone to data conflicts.

Once you have determined where the conflicts are most likely to occur, you need to determine how to resolve the conflict. For example, do you want the latest change to have precedence, or should one site over another have precedence?

As you read each of the sections describing the different conflict resolution methods, you will learn what each method is best suited for. So, read each section and then think about how your business would want to resolve any potential conflicts.

After you have identified the potential problem areas and have determined what business rules would resolve the problem, use Oracle's conflict resolution methods (or one of your own) to implement a conflict resolution system.

Create Conflict Resolution Methods for Update Conflicts

The most common data conflict occurs when the same row at two or more different sites were updated at the same time, or before the deferred transaction from one site was successfully propagated to the other sites.

One method to avoid update conflicts is to implement a synchronous replicated environment, though this solution requires large network resource.

The other solution is to use the Oracle conflict resolution methods to deal with update conflicts that may occur when the same row has received two or more updates.

Overwrite and Discard

The overwrite and discard methods ignore the values from either the originating or destination site and therefore can never guarantee convergence with more than one master site. These methods are designed to be used by a single master site and multiple snapshot sites, or with some form of a user-defined notification facility.

The overwrite method replaces the current value at the destination site with the new value from the originating site. Conversely, the discard method ignores the new value from the originating site.

> **See Also:** "ADD_conflicttype_RESOLUTION procedure" on page 8-78 and "Overwrite and Discard" in *Oracle8i Replication* for more information.

Note: This section uses objects not found in the other scripts within this book, because the configuration ORC1.WORLD, ORC2.WORLD, ORC3.WORLD, and SNAP1.WORLD contains three master sites and one snapshot site and is not appropriate for OVERWRITE and DISCARD.

```
--The following procedures need to be executed by the replication administrator.

CONNECT repadmin/repadmin@saturn.universe

--Before you can define any conflict resolution methods, quiesce the
--master group that contains the table to which you want to apply the
--conflict resolution method.

BEGIN
   DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
      GNAME => 'titan_mg');
END;
/

--All Oracle conflict resolution methods are based on logical column groupings
--called "column groups."  Create a column group for your target table by using
--the DBMS_REPCAT.MAKE_COLUMN_GROUP procedure.

BEGIN
   DBMS_REPCAT.MAKE_COLUMN_GROUP (
      SNAME => 'titan',
      ONAME => 'planet',
      COLUMN_GROUP => 'planet_cg1',
      LIST_OF_COLUMN_NAMES => 'order,circumference,moons');
END;
/

--Use the DBMS_REPCAT.ADD_UPDATE_RESOLUTION API to define the conflict
--resolution method for a specified table. This example creates an
--"Overwrite" conflict resolution method.

BEGIN
   DBMS_REPCAT.ADD_UPDATE_RESOLUTION (
      SNAME => 'titan',
      ONAME => 'planet',
      COLUMN_GROUP => 'planet_cg1',
      SEQUENCE_NO => 1,
      METHOD => 'OVERWRITE',
      PARAMETER_COLUMN_NAME => 'order,circumference,moons');
END;
/
```

```
--After you have defined your conflict resolution method, regenerate
--replication support for the table that received the conflict
--resolution method.

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      SNAME => 'titan',
      ONAME => 'planet',
      TYPE => 'TABLE',
      MIN_COMMUNICATION => TRUE);
END;
/

--After replication support has been regenerated, resume replication
--activity by using the RESUME_MASTER_ACTIVITY procedure API.

BEGIN
   DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
      GNAME => 'titan_mg');
END;
/
```

Minimum and Maximum

When the advanced replication facility detects a conflict with a column group and calls either the *minimum* or *maximum* value conflict resolution methods, it compares the new value from the originating site with the current value from the destination site for a designated column in the column group. You must designate this column when you define your conflict resolution method.

If the new value of the designated column is *less than* or *greater than* (depending on the method used) the current value, the column group values from the originating site are applied at the destination site, assuming that all other errors were successfully resolved for the row. Otherwise the rows remain unchanged.

```
--The following procedures need to be executed by the replication administrator.

CONNECT repadmin/repadmin@orc1.world

--Before you can define any conflict resolution methods, quiesce the
--master group that contains the table to which you want to apply the
--conflict resolution method.
```

```
BEGIN
   DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/

--All Oracle conflict resolution methods are based on logical column groupings
--called "column groups."  Create a column group for your target table by using
--the DBMS_REPCAT.MAKE_COLUMN_GROUP procedure.

BEGIN
   DBMS_REPCAT.MAKE_COLUMN_GROUP (
      SNAME => 'scott',
      ONAME => 'salgrade',
      COLUMN_GROUP => 'salgrade_cg1',
      LIST_OF_COLUMN_NAMES => 'losal');
END;
/

--Use the DBMS_REPCAT.ADD_UPDATE_RESOLUTION API to define the conflict
--resolution method for a specified table. This example creates a
--"MINIMUM" conflict resolution method.

BEGIN
   DBMS_REPCAT.ADD_UPDATE_RESOLUTION (
      SNAME => 'scott',
      ONAME => 'salgrade',
      COLUMN_GROUP => 'salgrade_cg1',
      SEQUENCE_NO => 1,
      METHOD => 'MINIMUM',
      PARAMETER_COLUMN_NAME => 'losal');
END;
/

--After you have defined your conflict resolution method, regenerate
--replication support for the table that received the conflict
--resolution method.
```

```
BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      SNAME => 'scott',
      ONAME => 'salgrade',
      TYPE => 'TABLE',
      MIN_COMMUNICATION => TRUE);
END;
/

--After replication support has been regenerated, resume replication
--activity by using the RESUME_MASTER_ACTIVITY procedure API.

BEGIN
   DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
      GNAME => 'scott_rg');
END;
/
```

Timestamp

The ***earliest timestamp*** and ***latest timestamp*** methods are variations on the minimum and maximum value methods. To use the timestamp method, you must designate a column in the replicated table of type DATE. When an application updates any column in a column group, the application must also update the value of the designated timestamp column with the local SYSDATE. For a change applied from another site, the timestamp value should be set to the timestamp value from the originating site.

Several elements are needed to make timestamp conflict resolution work well:

- Synchronized Time Settings Between Computers
- Timestamp field and trigger to automatically record timestamp

```
--The following procedures need to be executed by the replication administrator.

CONNECT repadmin/repadmin@orc1.world

--Before you can define any conflict resolution methods, quiesce the
--master group that contains the table to which you want to apply the
--conflict resolution method.
```

```
BEGIN
   DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/

--If the target table does not already contain a timestamp field,
--then add an additional column to your table to record the
--timestamp value when a row is inserted or updated. Additionally,
--you must use the ALTER_MASTER_REPOBJECT API to apply the DDL to
--the target table. Simply issuing the DDL may cause the replicated
--object to become invalid.

BEGIN
   DBMS_REPCAT.ALTER_MASTER_REPOBJECT (
      SNAME => 'scott',
      ONAME => 'emp',
      TYPE => 'TABLE',
      DDL_TEXT => 'ALTER TABLE scott.emp ADD (timestamp DATE)');
END;
/

--After you have inserted a new column into your replicated object,
--make sure that you re-generate replication support for the
--affected object. This step should be performed immediately
--after you alter the replicated object.

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      SNAME => 'scott',
      ONAME => 'emp',
      TYPE => 'TABLE',
      MIN_COMMUNICATION => TRUE);
END;
/

--Once the timestamp field has been created, create a trigger
--that records the timestamp of when a row is either inserted
--or updated. This recorded value is used in the resolution of
--conflicts based on the Timestamp method. Instead of directly executing the
--DDL, you should use the DBMS_REPCAT.CREATE_MASTER_REPOBJECT procedure to
--create the trigger and add it to your master group.
```

```
BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
      GNAME => 'scott_mg',
      TYPE => 'TRIGGER',
      ONAME => 'insert_time',
      SNAME => 'scott',
      DDL_TEXT => 'CREATE TRIGGER scott.insert_time
                     BEFORE
                        INSERT OR UPDATE ON scott.emp FOR EACH ROW
                     BEGIN
                        IF DBMS_REPUTIL.FROM_REMOTE = FALSE THEN
                           :NEW.TIMESTAMP := SYSDATE;
                        END IF;
                     END;');
END;
/

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      SNAME => 'scott',
      ONAME => 'insert_time',
      TYPE => 'TRIGGER',
      MIN_COMMUNICATION => TRUE);
END;
/

--All Oracle conflict resolution methods are based on logical column groupings
--called "column groups."  Create a column group for your target table by using
--the DBMS_REPCAT.MAKE_COLUMN_GROUP procedure.

BEGIN
   DBMS_REPCAT.MAKE_COLUMN_GROUP (
      SNAME => 'scott',
      ONAME => 'emp',
      COLUMN_GROUP => 'emp_cgl',
      LIST_OF_COLUMN_NAMES => 'mgr, hiredate, sal, timestamp');
END;
/
```

```
--Use the DBMS_REPCAT.ADD_UPDATE_RESOLUTION API to define the conflict
--resolution method for a specified table. This example specifies the
--"LATEST TIMESTAMP" conflict resolution method using the TIMESTAMP column
--that you created earlier.

BEGIN
   DBMS_REPCAT.ADD_UPDATE_RESOLUTION (
      SNAME => 'scott',
      ONAME => 'emp',
      COLUMN_GROUP => 'emp_cg1',
      SEQUENCE_NO => 1,
      METHOD => 'LATEST TIMESTAMP',
      PARAMETER_COLUMN_NAME => 'timestamp');
END;
/

--After you have defined your conflict resolution method, regenerate
--replication support for the table that received the conflict
--resolution method.

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      SNAME => 'scott',
      ONAME => 'emp',
      TYPE => 'TABLE',
      MIN_COMMUNICATION => TRUE);
END;
/

--After replication support has been regenerated, resume replication
--activity by using the RESUME_MASTER_ACTIVITY procedure API.

BEGIN
   DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/
```

Additive and Average

The *additive* and *average* methods work with column groups consisting of a single numeric column only. Instead of "accepting" one value over another, this conflict resolution method either adds the two compared values together or takes an average of the two compared values.

```
--The following procedures need to be executed by the replication administrator.

CONNECT repadmin/repadmin@orc1.world

--Before you can define any conflict resolution methods, quiesce the
--master group that contains the table to which you want to apply the
--conflict resolution method.

BEGIN
   DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/

--All Oracle conflict resolution methods are based on logical column groupings
--called "column groups."  Create a column group for your target table by using
--the DBMS_REPCAT.MAKE_COLUMN_GROUP procedure.

BEGIN
   DBMS_REPCAT.MAKE_COLUMN_GROUP (
      SNAME => 'scott',
      ONAME => 'bonus',
      COLUMN_GROUP => 'bonus_cg1',
      LIST_OF_COLUMN_NAMES => 'sal');
END;
/

--Use the DBMS_REPCAT.ADD_UPDATE_RESOLUTION API to define the conflict
--resolution method for a specified table. This example specifies the
--"ADDITIVE" conflict resolution method using the SAL column.
```

```
BEGIN
   DBMS_REPCAT.ADD_UPDATE_RESOLUTION (
      SNAME => 'scott',
      ONAME => 'bonus',
      COLUMN_GROUP => 'bonus_cg1',
      SEQUENCE_NO => 1,
      METHOD => 'ADDITIVE',
      PARAMETER_COLUMN_NAME => 'sal');
END;
/

--After you have defined your conflict resolution method, regenerate
--replication support for the table that received the conflict
--resolution method.

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      SNAME => 'scott',
      ONAME => 'bonus',
      TYPE => 'TABLE',
      MIN_COMMUNICATION => TRUE);
END;
/

--After replication support has been regenerated, resume replication
--activity by using the RESUME_MASTER_ACTIVITY procedure API.

BEGIN
   DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/
```

Priority Groups

Priority groups allow you to assign a priority level to each possible value of a particular column. If Oracle detects a conflict, Oracle updates the table whose "priority" column has a lower value using the data from the table with the higher priority value.

```
CONNECT repadmin/repadmin@orcl.world

BEGIN
   DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/

--Make sure that the JOB field is part of the column group that your
--site priority conflict resolution mechanism is used for. Use the
--ADD_GROUPED_COLUMN procedure to add this field to an existing column group.
--If you do not already have a column group, you can create a new column group
--using the DBMS_REPCAT.MAKE_COLUMN_GROUP procedure.

BEGIN
   DBMS_REPCAT.MAKE_COLUMN_GROUP (
      SNAME => 'scott',
      ONAME => 'emp',
      COLUMN_GROUP => 'emp_cg1',
      LIST_OF_COLUMN_NAMES => 'mgr, hiredate, sal, job');
END;
/

--Before you begin assigning a priority value to the values in your table, you
--must create a priority group that "holds" the values that you defined.

BEGIN
   DBMS_REPCAT.DEFINE_PRIORITY_GROUP (
      GNAME => 'scott_mg',
      PGROUP => 'job_pg',
      DATATYPE => 'VARCHAR2');
END;
/
```

```
--The DBMS_REPCAT.ADD_PRIORITY_datatype procedure is avaiable in several
--different versions. There is a version for each available datatype
--(NUMBER, VARCHAR2, and so on). See "ADD_PRIORITY_datatype procedure"
-- on page 8-75 for more information. Execute this API as often as
--necessary until you have defined a priority value for all possible
--table values.

BEGIN
   DBMS_REPCAT.ADD_PRIORITY_VARCHAR2(
      GNAME => 'scott_mg',
      PGROUP => 'job_pg',
      VALUE => 'president',
      PRIORITY => 100);
END;
/

BEGIN
   DBMS_REPCAT.ADD_PRIORITY_VARCHAR2(
      GNAME => 'scott_mg',
      PGROUP => 'job_pg',
      VALUE => 'manager',
      PRIORITY => 80);
END;
/

BEGIN
   DBMS_REPCAT.ADD_PRIORITY_VARCHAR2(
      GNAME => 'scott_mg',
      PGROUP => 'job_pg',
      VALUE => 'salesman',
      PRIORITY => 60);
END;
/

BEGIN
   DBMS_REPCAT.ADD_PRIORITY_VARCHAR2(
      GNAME => 'scott_mg',
      PGROUP => 'job_pg',
      VALUE => 'analyst',
      PRIORITY => 40);
END;
/
```

```
BEGIN
   DBMS_REPCAT.ADD_PRIORITY_VARCHAR2(
      GNAME => 'scott_rg',
      PGROUP => 'job_pg',
      VALUE => 'clerk',
      PRIORITY => 20);
END;
/

--After you have completed assigning your priority values, add the
--PRIORITY GROUP resolution method to your replicated table. The following API
--example shows that it is the second conflict resolution method for the
--specified column group (SEQUENCE_NO).

BEGIN
   DBMS_REPCAT.ADD_UPDATE_RESOLUTION (
      SNAME => 'scott',
      ONAME => 'emp',
      COLUMN_GROUP => 'emp_cg1',
      SEQUENCE_NO => 2,
      METHOD => 'PRIORITY GROUP',
      PARAMETER_COLUMN_NAME => 'job',
      PRIORITY_GROUP => 'job_pg');
END;
/

--After you have defined your conflict resolution method, regenerate
--replication support for the table that received the conflict
--resolution method.

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      SNAME => 'scott',
      ONAME => 'emp',
      TYPE => 'TABLE',
      MIN_COMMUNICATION => TRUE);
END;
/
```

```
--After replication support has been regenerated, resume replication
--activity by using the RESUME_MASTER_ACTIVITY procedure API.

BEGIN
   DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/
```

Site Priority

Site priority is a specialized form of priority groups. Therefore, many of the procedures associated with site priority behave similarly to the procedures associated with priority groups. Instead of resolving a conflict based on the priority of a field's value, the conflict is resolved based on the priority of the sites involved.

For example, if you assign ORC2.WORLD a higher priority value than ORC1.WORLD and a conflict arises between these two sites, the value from ORC2.WORLD is used.

```
CONNECT repadmin/repadmin@orc1.world

BEGIN
   DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/

--You must add a SITE column to your table to store the site value in
--your replicated table. Use the DBMS_REPCAT.ALTER_MASTER_REPOBJECT procedure
--to apply the DDL to the target table. Simply issuing the DDL may cause
--the replicated object to become invalid.

BEGIN
   DBMS_REPCAT.ALTER_MASTER_REPOBJECT (
      SNAME => 'scott',
      ONAME => 'emp',
      TYPE => 'TABLE',
      DDL_TEXT => 'ALTER TABLE scott.emp ADD (site VARCHAR2(20))');
END;
/
```

```
--After you have inserted a new column into your replicated object,
--make sure that you re-generate replication support for the
--affected object. This step should be performed immmediately
--after you alter the replicated object.

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      SNAME => 'scott',
      ONAME => 'emp',
      TYPE => 'TABLE',
      MIN_COMMUNICATION => TRUE);
END;
/

--After you have added the SITE column to your table, make sure
--that this field is part of the column group that your site
--priority conflict resolution mechanism is used for. Use the
--ADD_GROUPED_COLUMN procedure to add this field to an existing
--column group. If you do not already have a column group, you can create a
--new column group using the DBMS_REPCAT.MAKE_COLUMN_GROUP procedure.

BEGIN
   DBMS_REPCAT.MAKE_COLUMN_GROUP (
      SNAME => 'scott',
      ONAME => 'emp',
      COLUMN_GROUP => 'emp_cg1',
      LIST_OF_COLUMN_NAMES => 'mgr, hiredate, sal, site');
END;
/

--Before you begin assigning a site priority value to the sites in your
--replicated environment, you must create a site priority group that "holds"
--the values that you defined.

BEGIN
   DBMS_REPCAT.DEFINE_SITE_PRIORITY (
      GNAME => 'scott_rg',
      NAME => 'site_pg');
END;
/

--Define the priority value for each of the sites in your replication
--environment using the DBMS_REPCAT.ADD_SITE_PRIORITY_SITE procedure.
--Execute this API as often as necessary until you have defined a site
--priority value for each of the sites in our replication environment.
```

```
BEGIN
   DBMS_REPCAT.ADD_SITE_PRIORITY_SITE (
      GNAME => 'scott_mg',
      NAME => 'site_pg',
      SITE => 'orc1.world',
      PRIORITY => 100);
END;
/

BEGIN
   DBMS_REPCAT.ADD_SITE_PRIORITY_SITE (
      GNAME => 'scott_mg',
      NAME => 'site_pg',
      SITE => 'orc2.world',
      PRIORITY => 50);
END;
/

BEGIN
   DBMS_REPCAT.ADD_SITE_PRIORITY_SITE (
      GNAME => 'scott_mg',
      NAME => 'site_pg',
      SITE => 'orc3.world',
      PRIORITY => 25);
END;
/

--After you have completed assigning your site priority values, add the
--SITE PRIORITY resolution method to your replicated table. The following
--API examples shows that it is the third conflict resolution method
--for the specified column group (SEQUENCE_NO).

BEGIN
   DBMS_REPCAT.ADD_UPDATE_RESOLUTION (
      SNAME => 'scott',
      ONAME => 'emp',
      COLUMN_GROUP => 'emp_cg1',
      SEQUENCE_NO => 3,
      METHOD => 'site priority',
      PARAMETER_COLUMN_NAME => 'site',
      PRIORITY_GROUP => 'site_pg');
END;
/
```

```
--After you have defined your conflict resolution method, regenerate
--replication support for the table that received the conflict
--resolution method.

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      SNAME => 'scott',
      ONAME => 'emp',
      TYPE => 'TABLE',
      MIN_COMMUNICATION => TRUE);
END;
/

--After replication support has been regenerated, resume replication
--activity by using the RESUME_MASTER_ACTIVITY procedure API.

BEGIN
   DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/
```

Create Conflict Resolution Methods for Uniqueness Conflicts

In a replicated environment, you may encounter situations where you receive a conflict on a unique constraint, often resulting from an insert. If your business rules allow you to delete the duplicate row, you can define such resolution with Oracle's pre-built conflict resolution methods.

More often, however, you probably want to modify the conflicting value so that it no longer violates the unique constraint. Modifying the conflicting value ensures that you do not lose important data. Oracle's pre-built uniqueness conflict resolution method can make the conflicting value unique by appending a site name or a sequence number to the value.

An additional component that accompanies uniqueness conflict resolution methods is a notification facility. The conflicting information is modified by Oracle so that it can be inserted into the table, but you should be notified so that you can analyze the conflict to determine whether the record should be deleted, or the data merged into another record, or a completely new value be defined for the conflicting data.

```
--The following procedures need to be executed by the replication administrator.

CONNECT repadmin/repadmin@orc1.world

BEGIN
   DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/

--As you might expect, a uniqueness conflict resolution method detects and
--resolves conflicts encountered on columns with a UNIQUE constraint. Use
--the ALTER_MASTER_REPOBJECT procedure (described on page 8-83) to add
--a UNIQUE constraint to the EMP table.

BEGIN
   DBMS_REPCAT.ALTER_MASTER_REPOBJECT (
      SNAME => 'scott',
      ONAME => 'emp',
      TYPE => 'TABLE',
      DDL_TEXT => 'ALTER TABLE scott.emp ADD
                  (constraint emp_ename_unique UNIQUE(ename))');
END;
/

--After you have added the UNIQUE constraint to your replicated table,
--make sure that you regenerate replication support for
--the affected table. This step should be performed immediately
--after you alter the replicated object.

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      SNAME => 'scott',
      ONAME => 'emp',
      TYPE => 'TABLE',
      MIN_COMMUNICATION => TRUE);
END;
/
```

```
--The following table (conf_report) stores the messages received from
--your notification facility.

BEGIN
   DBMS_REPCAT.EXECUTE_DDL(
      GNAME => 'scott_mg',
      DDL_TEXT => 'CREATE TABLE scott.conf_report (
                   line NUMBER(2),
                   txt VARCHAR2(80),
                   timestamp DATE,
                   table_name VARCHAR2(30),
                   table_owner VARCHAR2(30),
                   conflict_type VARCHAR2(7))');
END;
/

CONNECT scott/tiger@orc1.world

--The following package (notify) sends a notification to the CONF_REPORT
--table when a conflict is detected.

--The conflict resolution notification package that is created in this script is
--described in detail in Appendix B, "User-Defined Conflict Resolution Methods".

CREATE OR REPLACE PACKAGE notify AS
   FUNCTION emp_unique_violation (ename IN OUT VARCHAR2,
      discard_new_values IN OUT BOOLEAN)
   RETURN BOOLEAN;
END notify;
/
```

```
CREATE OR REPLACE PACKAGE BODY notify AS
   TYPE message_table IS TABLE OF VARCHAR2(80) INDEX BY BINARY_INTEGER;
   PROCEDURE report_conflict(conflict_report IN MESSAGE_TABLE,
      report_length IN NUMBER,
      conflict_time IN DATE,
      conflict_table IN VARCHAR2,
      table_owner IN VARCHAR2,
      conflict_type IN VARCHAR2) IS
      BEGIN
         FOR idx IN 1..report_length LOOP
            BEGIN
               INSERT INTO scott.conf_report
               (line, txt, timestamp, table_name, table_owner, conflict_type)
               VALUES (idx, SUBSTR(conflict_report(idx),1,80), conflict_time,
               conflict_table, table_owner, conflict_type);
            EXCEPTION WHEN others THEN NULL;
            END;
         END LOOP;
      END report_conflict;
   FUNCTION emp_unique_violation(ename IN OUT VARCHAR2,
      discard_new_values IN OUT BOOLEAN)
   RETURN BOOLEAN IS
      local_node VARCHAR2(128);
      conf_report MESSAGE_TABLE;
      conf_time DATE := SYSDATE;
   BEGIN
      BEGIN
         SELECT global_name INTO local_node FROM global_name;
      EXCEPTION WHEN others THEN local_node := '?';
      END;
      conf_report(1) := 'UNIQUENESS CONFLICT DETECTED IN TABLE EMP ON ' ||
         TO_CHAR(conf_time, 'MM-DD-YYYY HH24:MI:SS');
      conf_report(2) := ' AT NODE ' || local_node;
      conf_report(3) := 'ATTEMPTING TO RESOLVE CONFLICT USING' ||
         ' APPEND SITE NAME METHOD';
      conf_report(4) := 'ENAME: ' || ename;
      conf_report(5) := NULL;
      report_conflict(conf_report, 5, conf_time, 'EMP', 'SCOTT', 'UNIQUE');
      discard_new_values := FALSE;
      RETURN FALSE;
   END emp_unique_violation;
END notify;
/
```

```
CONNECT repadmin/repadmin@orcl.world

--The following package is replicated to all of the master sites in your
--replication environment, which ensures that the notification facility is
--available at all master sites.

BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
      GNAME => 'scott_mg',
      TYPE =>  'PACKAGE',
      ONAME => 'notify',
      SNAME => 'scott');
END;
/

BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
      GNAME => 'scott_mg',
      TYPE =>  'PACKAGE BODY',
      ONAME => 'notify',
      SNAME => 'scott');
END;
/

--After you have completed building your notification facility, add the
--notification facility as one of your conflict resolution methods,
--even though it only notifies of a conflict. The following API example
--demonstrates adding the notification facility as a USER FUNCTION.

BEGIN
   DBMS_REPCAT.ADD_UNIQUE_RESOLUTION(
      SNAME => 'scott',
      ONAME => 'emp',
      CONSTRAINT_NAME => 'emp_ename_unique',
      SEQUENCE_NO => 1,
      METHOD => 'USER FUNCTION',
      COMMENT => 'Notify DBA',
      PARAMETER_COLUMN_NAME => 'ename',
      FUNCTION_NAME => 'scott.notify.emp_unique_violation');
END;
/
```

```
--After you have added the notification facility, you are ready to add the
--actual conflict resolution method to your table. The following API example
--demonstrates adding the APPEND SITE NAME uniqueness conflict resolution
--method to your replicated table.

BEGIN
   DBMS_REPCAT.ADD_UNIQUE_RESOLUTION(
      SNAME => 'scott',
      ONAME => 'emp',
      CONSTRAINT_NAME => 'emp_ename_unique',
      SEQUENCE_NO => 2,
      METHOD => 'APPEND SITE NAME',
      PARAMETER_COLUMN_NAME => 'ename');
END;
/

--After you have defined your conflict resolution methods, regenerate
--replication support for the table that received the conflict
--resolution methods.

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      SNAME => 'scott',
      ONAME => 'emp',
      TYPE => 'TABLE',
      MIN_COMMUNICATION => TRUE);
END;
/

--After replication support has been regenerated, resume replication
--activity by using the RESUME_MASTER_ACTIVITY procedure API.

BEGIN
   DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/
```

Create Conflict Avoidance Methods for Delete Conflicts

Unlike update conflicts, where there are two values to compare, simply deleting a row makes the update conflict resolution methods described in the previous section ineffective because only one value would exist.

The best way to deal with deleting rows in a replication environment is to "avoid" the conflict by marking a row for deletion and periodically purging the table of all "marked" records. Because you are not physically removing this row, your data can converge at all master sites if a conflict arises because you still have two values to compare, assuming that no other errors have occurred. After you are sure that your data has converged, you can purge "marked" rows using a replicated purge procedure.

When you are developing your front-end application for your database, you probably want to "filter out" the rows that have been marked for deletion, because doing so makes it appear to your users as though the row was physically deleted. Simply exclude the rows that have been marked for deletion in the SELECT statement for your data set. For example, a select statement for a current employee listing might be similar to the following:

```
SELECT * FROM emp WHERE remove_date IS NULL;
```

This section describes how to prepare your replicated table to avoid delete conflicts. You also see how to use procedural replication to purge those records that have been "marked" for deletion.

```
CONNECT repadmin/repadmin@orcl.world

BEGIN
   DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/
```

```
--You must add a column to your replicated table that stores the
--mark for deleted records. It is advisable to use a timestamp to mark your
--records for deletion (timestamp reflects when the record was marked for
--deletion). Because you are using a timestamp, your new column must be
--a DATE datatype.  Use the DBMS_REPCAT.ALTER_MASTER_REPOBJECT procedure to add
--the REMOVE_DATE column to your existing replicated table.

BEGIN
   DBMS_REPCAT.ALTER_MASTER_REPOBJECT (
      SNAME => 'scott',
      ONAME => 'emp',
      TYPE => 'TABLE',
      DDL_TEXT => 'ALTER TABLE scott.emp ADD (remove_date DATE)');
END;
/

--After you have inserted a new column into your replicated object,
--make sure that you regenerate replication support for
--the affected object. This step should be performed immediately
--after you alter the replicated object.

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      SNAME => 'scott',
      ONAME => 'emp',
      TYPE => 'TABLE',
      MIN_COMMUNICATION => TRUE);
END;
/
```

```
--The following package is replicated to all of the master sites in your
--replication environment. This package purges all "marked" records from
--the specified table.

BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
      GNAME => 'scott_mg',
      TYPE => 'PACKAGE',
      ONAME => 'purge',
      SNAME => 'scott',
      DDL_TEXT => 'CREATE OR REPLACE PACKAGE scott.purge AS
                      PROCEDURE remove_emp(purge_date DATE);
                   END;');
END;
/

BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
      GNAME => 'scott_mg',
      TYPE => 'PACKAGE BODY',
      ONAME => 'purge',
      SNAME => 'scott',
      DDL_TEXT => 'CREATE OR REPLACE PACKAGE BODY scott.purge AS
                      PROCEDURE remove_emp(purge_date IN DATE) IS
                      BEGIN
                         DBMS_REPUTIL.REPLICATION_OFF;
                         LOCK TABLE scott.emp IN EXCLUSIVE MODE;
                         DELETE scott.emp WHERE remove_date IS NOT NULL AND
                            remove_date < purge_date;
                         DBMS_REPUTIL.REPLICATION_ON;
                      EXCEPTION WHEN others THEN
                         DBMS_REPUTIL.REPLICATION_ON;
                      END;
                   END;');
END;
/
```

```
--After you have created your package (package and package body), generate
--replication support for each component. After you generate
--replication support, a synonym is created for you and added to your
--master group as a replicated object. This synonym is labeled as
--DEFER_PURGE.REMOVE_EMP.

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      SNAME => 'scott',
      ONAME => 'purge',
      TYPE => 'PACKAGE',
      MIN_COMMUNICATION => TRUE);
END;
/

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      SNAME => 'scott',
      ONAME => 'purge',
      TYPE => 'PACKAGE BODY',
      MIN_COMMUNICATION => TRUE);
END;
/

--After replication support has been regenerated, resume replication
--activity by using the RESUME_MASTER_ACTIVITY procedure API.

BEGIN
   DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/
```

Audit Successful Conflict Resolution

Whenever Oracle detects and successfully resolves an update, delete, or uniqueness conflict, you can view information about what method was used to resolve the conflict by querying the DBA_REPRESOLUTION_STATISTICS data dictionary view. This view is updated only if you have chosen to turn on conflict resolution statistics gathering for the table involved in the conflict.

> **See Also:** The ALL_REPRESOLUTION_STATISTICS on page 9-32 for more information.

Gathering Conflict Resolution Statistics

Use the REGISTER_STATISTICS procedure in the DBMS_REPCAT package to collect information about the successful resolution of update, delete, and uniqueness conflicts for a table. The following example gathers statistics for the EMP table in the ACCT_REC schema:

```
DBMS_REPCAT.REGISTER_STATISTICS(sname     =>   'acct_rec',
                                oname     =>   'emp');
```

> **See Also:** The REGISTER_STATISTICS procedure on page 8-140 for more information.

Viewing Conflict Resolution Statistics

After you call REGISTER_STATISTICS for a table, each conflict that is successfully resolved for that table is logged in the DBA_REPRESOLUTION_ STATISTICS view. Information about unresolved conflicts is always logged to the DEFERROR view, whether the object is registered or not.

> **See Also:** ALL_REPRESOLUTION_STATISTICS on page 9-32 and DEFERROR on page 9-49 for more information.

Canceling Conflict Resolution Statistics

Use the CANCEL_STATISTICS procedure in the DBMS_REPCAT package if you no longer want to collect information about the successful resolution of update, delete, and uniqueness conflicts for a table. The following example cancels statistics gathering on the EMP table in the ACCT_REC schema:

```
DBMS_REPCAT.CANCEL_STATISTICS(sname     => '  acct_rec',
                              cname     =>   'emp');
```

See Also: The CANCEL_STATISTICS procedure on page 8-91 for more information.

Deleting Statistics Information

If you registered a table to log information about the successful resolution of update, delete, and uniqueness conflicts, you can remove this information from the DBA_REPRESOLUTION_STATISTICS view by calling the PURGE_STATISTICS procedure in the DBMS_REPCAT package.

The following example purges the statistics gathered about conflicts resolved due to inserts, updates, and deletes on the EMP table between January 1 and March 31:

```
DBMS_REPCAT.PURGE_STATISTICS(sname      =>   'acct_rec',
                             cname      =>   'emp',
                             start_date =>   '01-JAN-99',
                             end_date   =>   '31-MAR-99');
```

See Also: The PURGE_STATISTICS procedure on page 8-136 for more information.